Report Number: WUCS-85-05

1985-05-01

Hierarchical Dataflow Model: A Computation Model for School Children

Takayuki Dan Kimura

A new computation model suitable for icon based programming languages is proposed. The model is used to design a programming language for school children on the Macintosh personal computer. The model consists of boxes and arrows forming a partially ordered set of nested boxes. Loops and Boolean data tokens are eliminated from the traditional dataflow model. Block structures and logical consistency (exception) are added. Using a formal definition of the model it is proved that the model is determinate.

Hierarchical Dataflow Model:
A Computation Model for School Children

Takayuki Dan Kimura

WUCS-85-05

May 1985

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

This research was funded by Computer Services Corporation (CSK) of Japan.

Abstract

A new computation model suitable for icon based programming languages is proposed. The model is used to design a programming language for school children on the Macintosh personal computer. The model consists of boxes and arrows forming a partially ordered set of nested boxes. Loops and Boolean data tokens are eliminated from the traditional dataflow model. Block structures and logical consistency (exception) are added. Using a formal definition of the model it is proved that the model is determinate.

1. Introduction

This report introduces a computation model based on the concepts of dataflow and completion. The primary purpose of the model is to provide a semantic foundation for a new programming language called Show & Tell Language (STL) which was designed for school children. STL is a an icon driven programming language[1] that integrates the computer capabilities for managing computation, communication, and database in home and classroom environments. The design philosophy of the language and its implementation on Apple® Macintosh™ will be described in separate reports[2]. This report is self-contained and requires no knowledge of STL.

Dataflow[3] was chosen for its understandability. Concurrency was secondary. There are two factors that make it easy to understand a dataflow language; two dimensional representation and value oriented computation. Due to the principle of direct object manipulation[4], a graphical object is more amiable to school children than a textual one. The notion of variable, or equivalently the notion of state, is one of the most difficult concepts that school children have to learn for conventional Von Neumann type programming. Absence of variables in a dataflow language would make learning easier.

Dataflow also provides a good paradigm for message based communication facilities common in most home and classroom environments. A data-driven asynchronous execution of an operation is a typical mode of interaction among school children through communication media such as paper notes, telephone, and video. Dataflow is a natural integration mechanism for computation and communication.

A completion problem[5] is to fill missing portions of a partially hidden pattern in such a way that the completed pattern satisfies a set of consistency rules. Some examples of general completion problems are given in Figure 1.

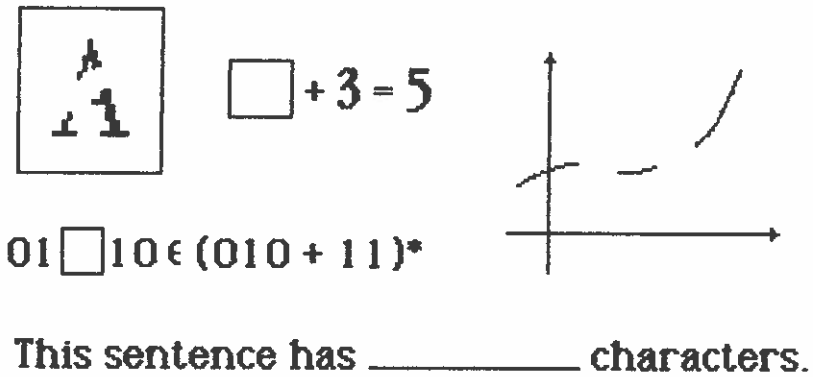
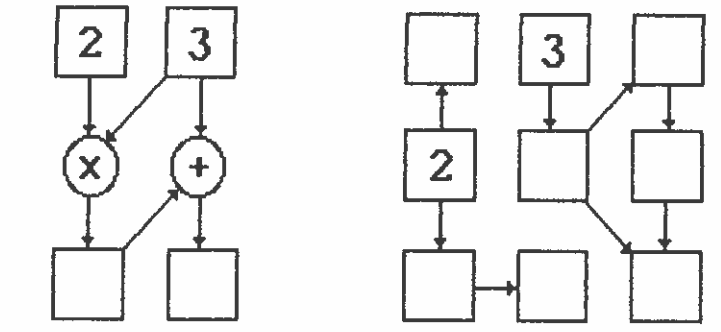
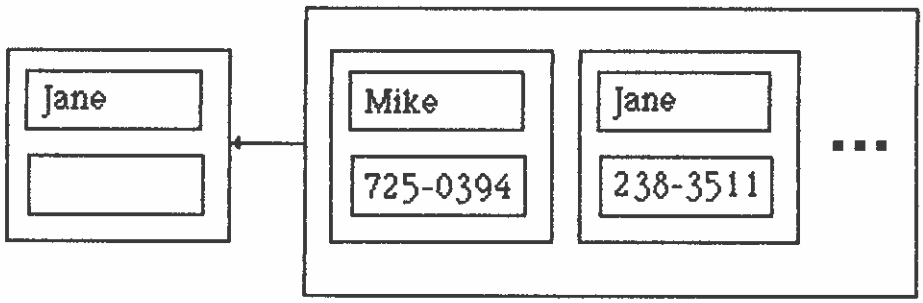


Figure 1 : General Completion Problem

Completion was chosen for its capability of integrating the notions of computation, communication and data query. A computation is a process to solve a completion problem whose consistency rules are arithmetic (Figure 2(a)). A communication is a process to complete the receivers with the exactly same information as the sender has. The consistency rules for such completion are defined by whose information to be shared with whom (Figure 2(b)). A data query is a completion problem where consistency rules are defined by a set of records in a file (Figure 2(c)). The computational completion problem can be solved by execution of arithmetic operations, the communication problem by transmission of messages, and the query completion problem by execution of pattern matching operations. Thus, completion is an unifying concept for computation, communication and data query. The Show & Tell language is designed as a specification language for such completion problems.



(a) Computation (b) Communication



(c) Data Query

Figure 2 : Completion as Integration Mechanism

The dataflow model by itself is not an appropriate computation model for school children. First of all, a cycle (loop) in a dataflow graph requires an introduction of the notion of state, and makes learning dataflow semantics difficult for children. Secondly, there is no abstraction mechanism inherent in the model by which complex problems can be decomposed into simpler ones. By the same token, there is no structuring construct similar to the begin-end block of ALGOL that is well-known for its effectiveness in program structuring. Thirdly, the switching operations, such as distributors and selectors, are not high level enough for children who are not familiar with hardware related concepts. A data token of a Boolean value representing the result of a decision making is counter-intuitive. Integration of control flow

with data flow through a data token of Boolean value is not appropriate for novice programmers. Finally, there is no encapsulation mechanism by which exception propagation can be controlled and modular programming can be exercised.

In order to resolve the above difficulties, a new computation model was developed by introducing the concepts of block structure and logical consistency into the traditional dataflow model. We call the model a hierarchical dataflow model (HDM). In the next section we will describe the model informally with examples using the STL syntax. The formal definition will be given subsequently. Then, the determinancy of the new model will be shown.

2. Boxes and Arrows

In the hierarchical dataflow model (HDM) a computation is specified by a box-graph, a set of boxes connected by a set of arrows. No cycle or loop is allowed in a box-graph. Each box may be empty or may contain a datum, an operation, a predicate, or another box-graph. An arrow directs the flow of data from one box to another. A box-graph is consistent if there is no conflict, directly or indirectly, among the contents of the boxes; otherwise inconsistent. For example, a box-graph is inconsistent if there are two boxes containing two different data values and directly connected by an arrow . An arrow between the two boxes defines the consistency relationship as well as data flow. Each arrow has exactly one starting box and one destination box. Arrows do not branch out. A copy of data value in the starting box of an arrow moves to the destination box, unless the destination box already contains a different data value. If it does, then the entire box-graph containing these boxes is inconsistent.

The set of empty boxes in a box-graph is called the base of the graph. A computation in HDM is to fill the base of a box-graph with a set of data in such a way that the completed graph is consistent. Note that higher level objects such as operations and predicates can not be used to fill empty boxes in HDM. Figure 3 (a) gives a simple computation problem consisting of five independent components. Figure 3 (b) displays a solution obtained by the Macintosh STL system. All inconsistent box-graphs are shaded by the system when solutions are derived. Note that the arithmetic operations and predicates in STL are system defined box-graphs with system defined names, therefore they can be either consistent or inconsistent.

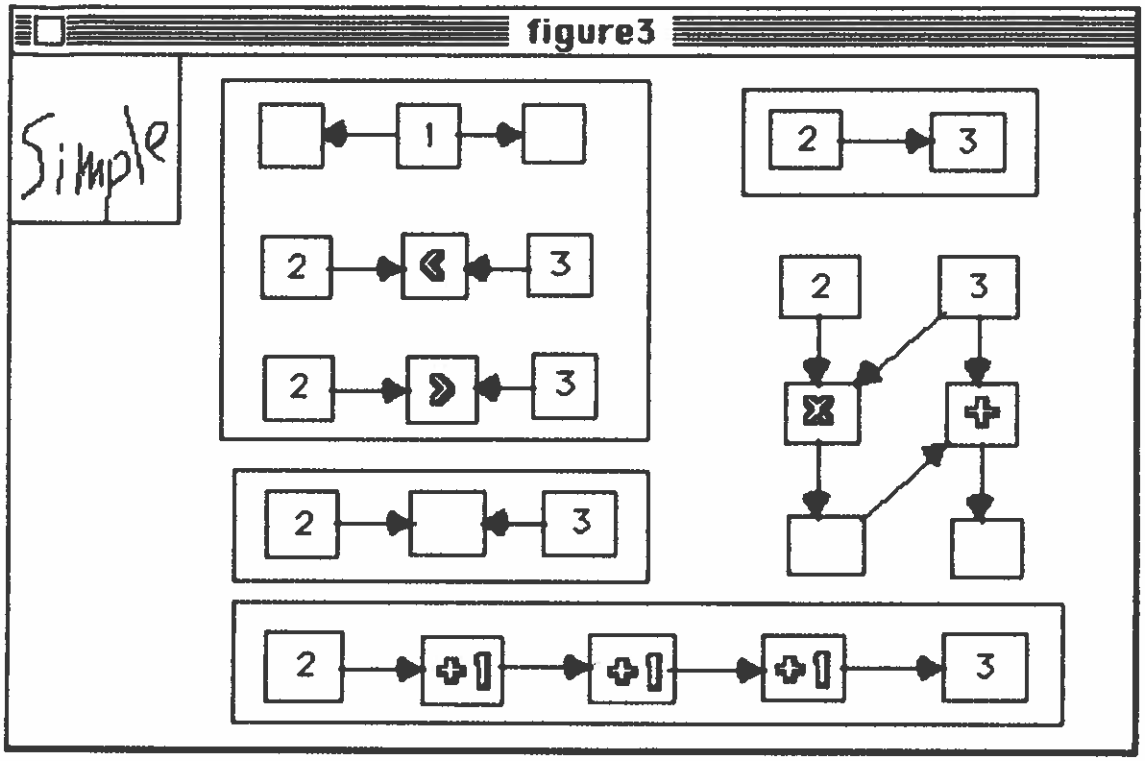


Figure 3: (a) Simple Box-Graph before Execution

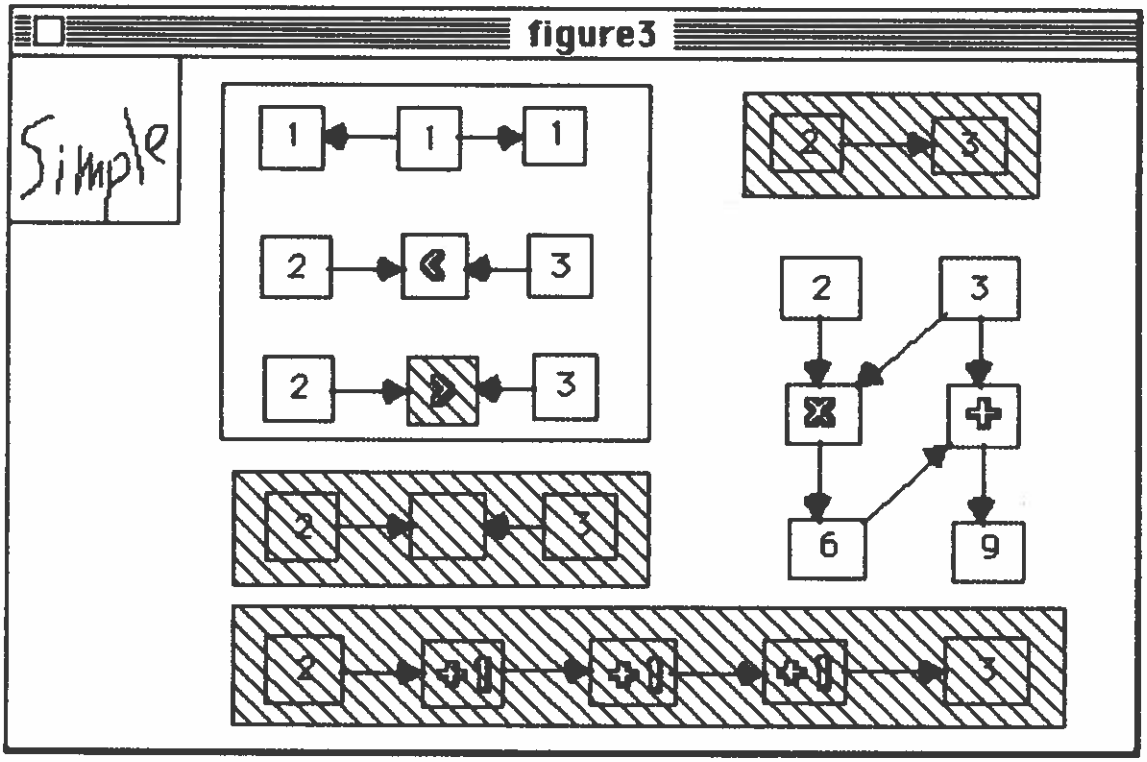


Figure 3: (b) Simple Box-Graph after Execution

Figure 3 was obtained by direct screen dump from the STL system. The square on the left-upper corner contains a user defined name (icon) of the box-graph drawn inside the window. An icon name stands for a box-graph when it is included in a box. Such an icon name corresponds to a subroutine name in traditional programming.

A box containing a box-graph is called a complex box. A complex box can be open, represented in STL by a dash-lined rectangle, or closed, represented by a solid-lined rectangle. If an open box contains an inconsistent box-graph, then the box-graph to which the open box belongs is considered to be inconsistent, i.e., inconsistency propagates through the boundary of an open box to the larger context. If a closed box contains an inconsistent box-graph, then the closed box becomes non-existent, i.e., the box with its content and all arrows incident with the box can be deleted from the box-graph without changing the graph's consistency property. In a sense, an open box broadcasts the exceptional condition (i.e., inconsistency) of its components towards the members of its community, and a closed box delimits the boundary of such broardcasting. This mode of communication, or propagation of information, is in contrast with that of point-to-point communication represented by an arrow.

Figure 4 with Figure 3 illustrates the differences between an open box and a closed box. In Figure 4, the left-upper component is inconsistent, because 2 is not greater than 3, and » is in the open box. The right-upper component is inconsistent, because there is a conflict in "2's flowing into 3" and the conflicting part is enclosed in an open box.

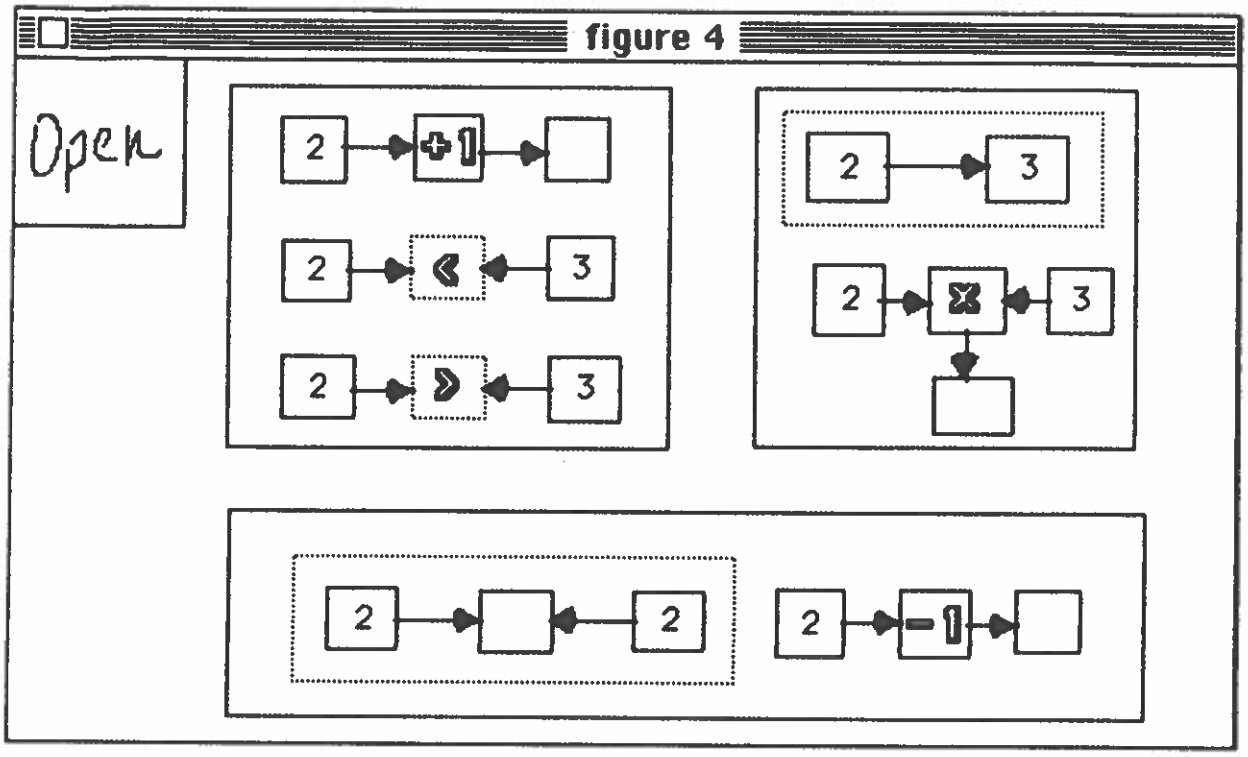


Figure 4: (a) Open Boxes before Execution

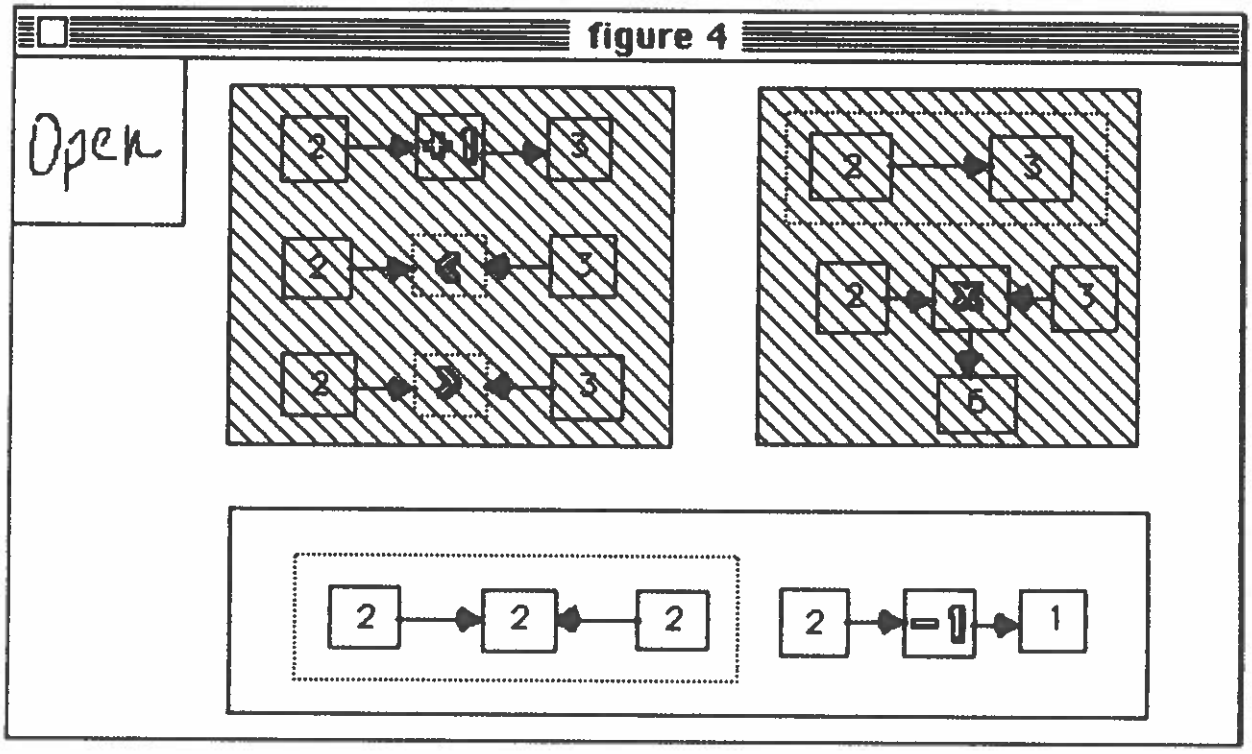
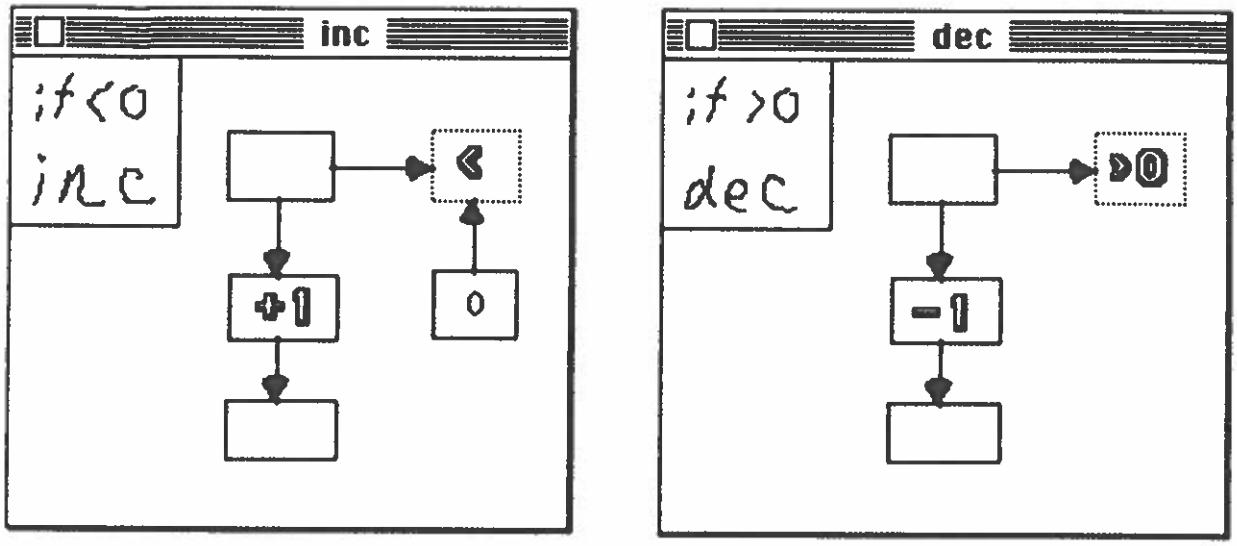


Figure 4: (b) Open Boxes after Execution

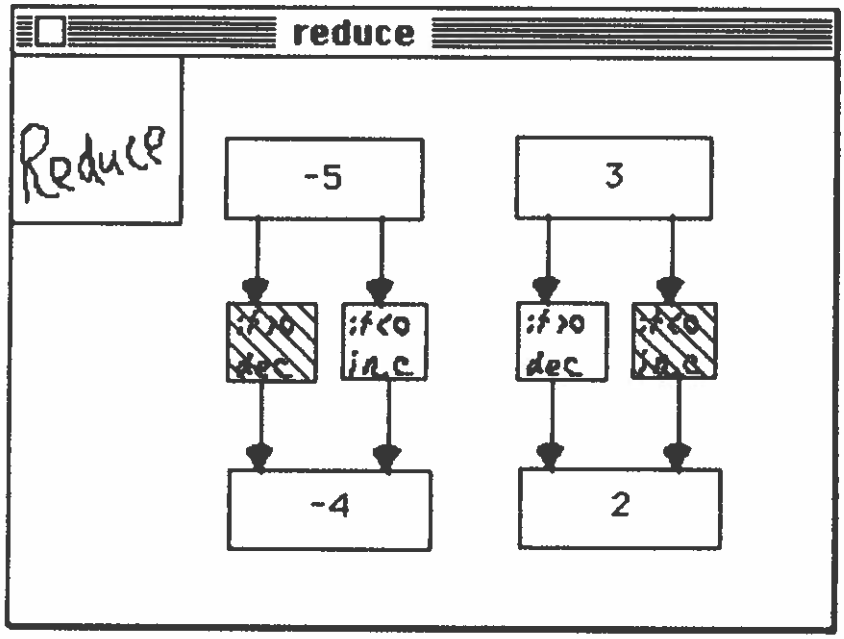
A box-graph defines a function of a traditional programming language. Since no cycle is allowed in a box-graph, every box-graph must contain at least one component box to which no arrow points and at least one box from which no arrow starts. These boxes are called the minimums and the maximums of the graph. The empty minimums and maximums are called the in-boxes and the out-boxes of the box-graph, respectively. They represent the input and the output parameters of the function. When the box-graph with all in-boxes filled by the argument data is solved, the out-boxes produce the value of the function. If there is no solution, the initial box-graph will be returned as inconsistent. In other words, a box-graph, representing a function, can return an exception condition as well as its value.

For example, in Figure 5 (a), the function **inc** returns the input value plus one if the input is negative, else it returns the exception (inconsistency). Similarly **dec** in Figure 5 (b) returns the exception if the input is not positive. Figure 5 (c) demonstrates how these functions can be incorporated in another function. If the box-graph does not have any out-box, it defines a predicate. When and only when the predicate is satisfied by the input arguments, the graph will be returned as a consistent graph.

A positional binding rule associates the incoming (outgoing) arrows to (from) a complex box with the in-boxes (out-boxes) of its content. The binding rule is language dependent. The model assumes that all arrows incident with a complex box can be ordered by some language specific rule. Similarly, the in-boxes and out-boxes of a box-graph can be ordered by another rule.



(a) (b)



(c)

Figure 5: Function with Exception

The first incoming arrow will be associated with the first in-box, the second incoming arrow with the second in-box, and so on. The same holds for binding the outgoing arrows with the out-boxes.

If the number of incoming arrows is different from the number of in-boxes, then the box-graph will be evaluated as inconsistent. The same is true for the outgoing arrows. In the STL system, the lexicographical ordering of the (x,y) coordinate on the Macintosh screen is used for ordering boxes and arrows. The coordinate of a box is defined as that of the left-upper corner of the box, and the coordinate of an arrow incident with a complex box is defined as that of the intersection point with the box. The binding rule of STL is illustrated by Figure 6 (a). Figure 6 (b) presents a STL convention that allows an arrow to cross the boundary of a complex box in order to make a binding explicit. Figure 6 (c) present a similar convention for an arrow intersecting with an empty box.

In summary, we will illustrate the notions introduced in this section by a sequence of examples. We will construct, first, a set of Boolean operations, then a half-adder for binary digits, followed by a 4 bit full-adder. No arithmetic operations will be used in the entire construction.

Figure 7.1 defines a standard set of Boolean operations. The name icons are chosen from the standard symbols for corresponding gate circuits. The boolean values are represented by integers 1 and 0. Figure 7 (a) through (h), except for (c), give definitions of the boolean operations in STL. Figure 7 (c) is the result of solving (a) with the inputs 1 and 0, and demonstrates how **and** operation works.

Figure 7 (i) defines a half-adder using the previously defined functions (subroutines), **and** and **exclusive-or**. It has three inputs x, y, and c (carry) from the left, and two outputs, c (next carry) and s (sum). Figure 7 (j) is a construction of a full-adder of length 4 using the half-adder of (i), and for testing the construction, the input value 0101 is given for x and 0111 for y, to generate the output 1100 for the sum. Finally, the full-adder is tested as an independent module in Figure 7 (k).

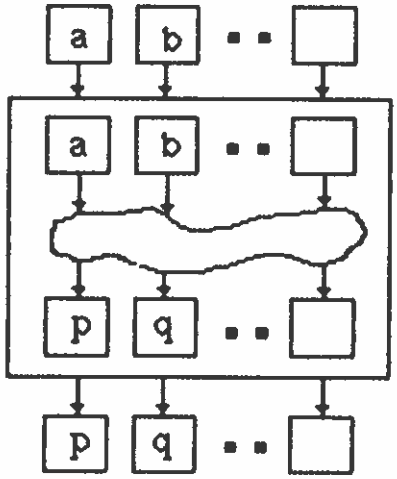


Figure 6 : (a) STL Binding Rule

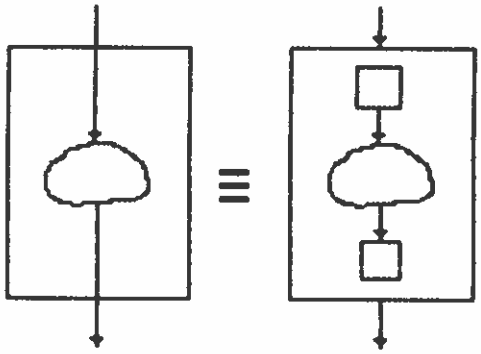


Figure 6 : (b) STL Convention 1

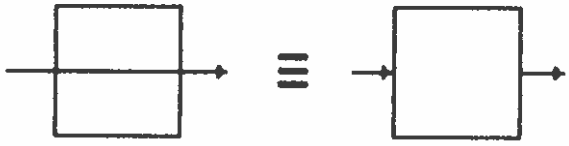
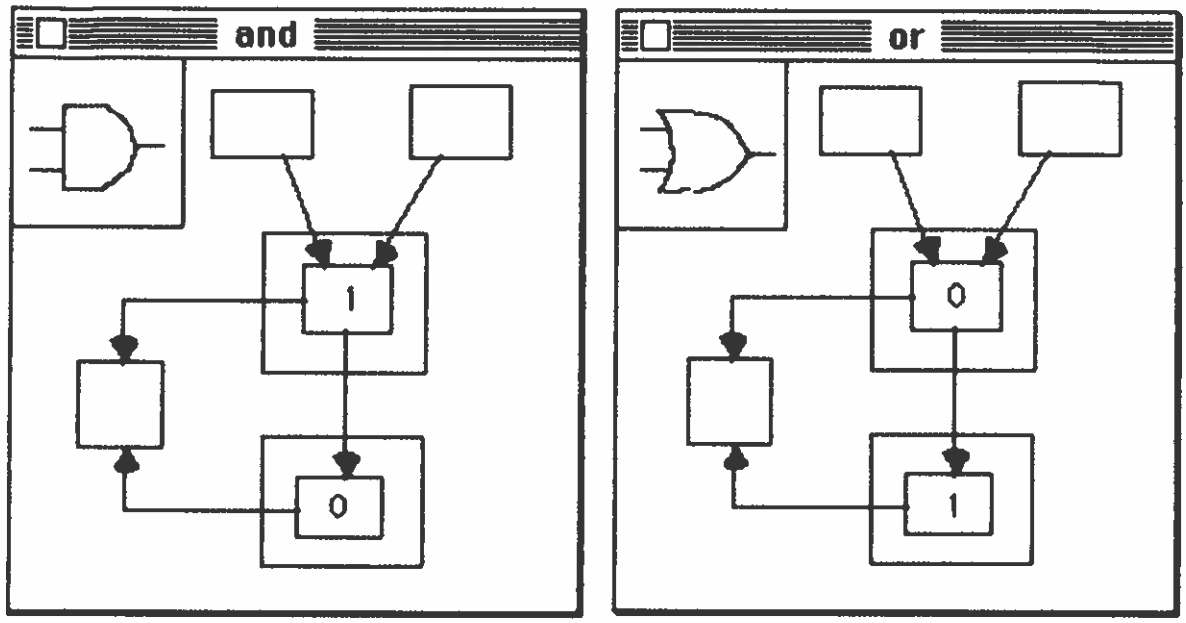


Figure 6 : (c) STL Convention 2



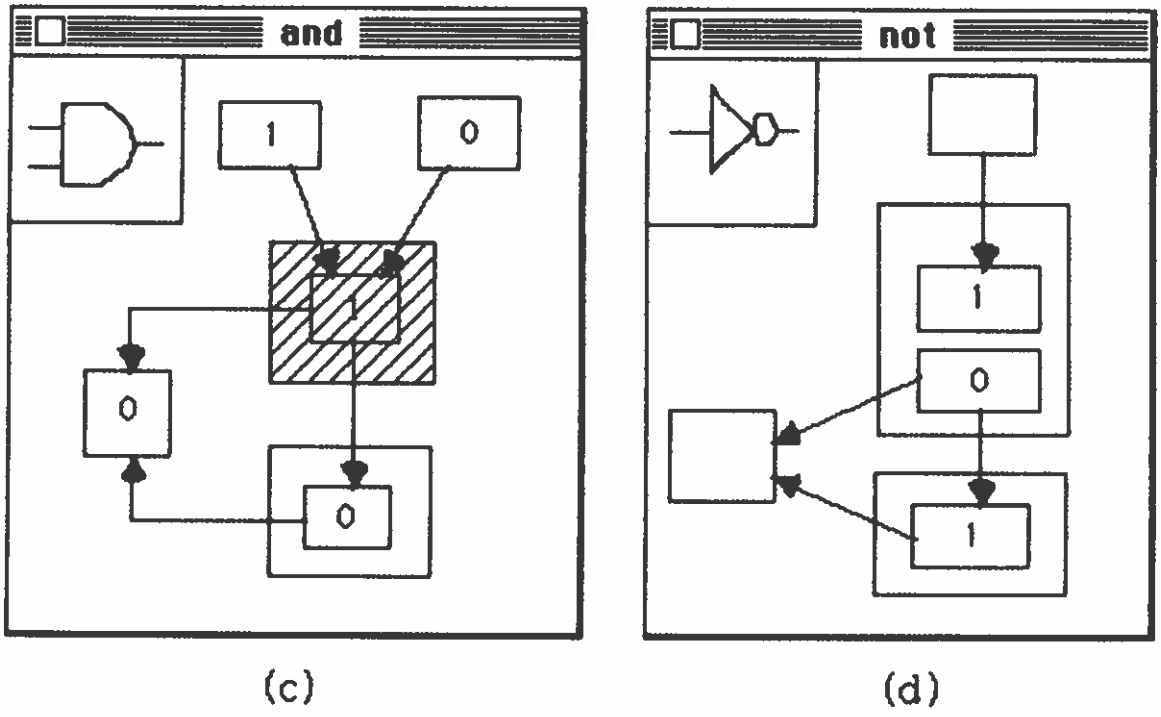
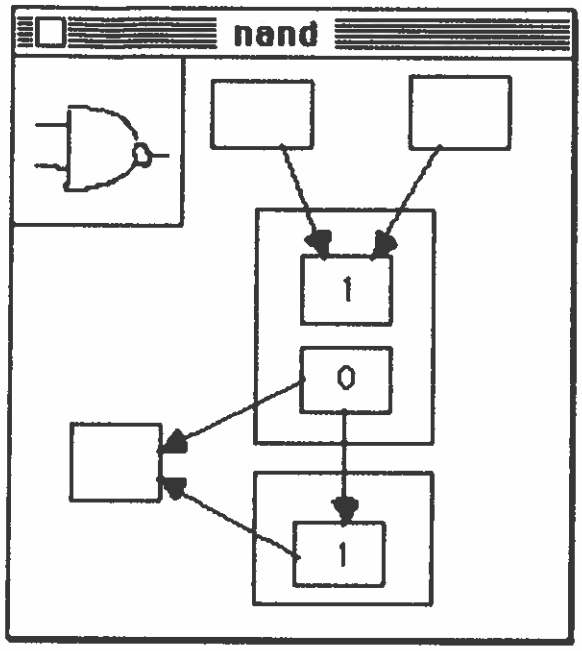
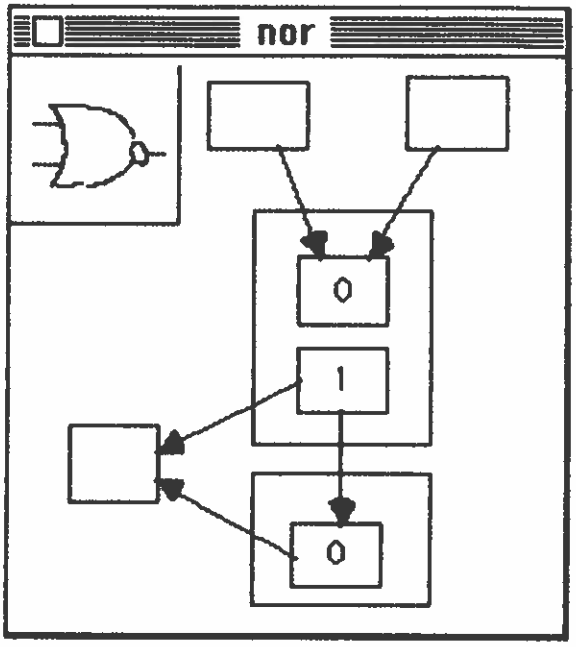


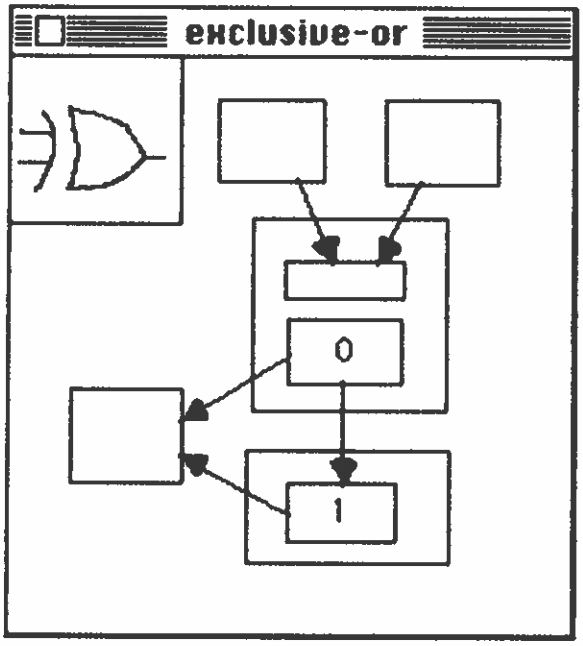
Figure 7.1 : Boolean Operations



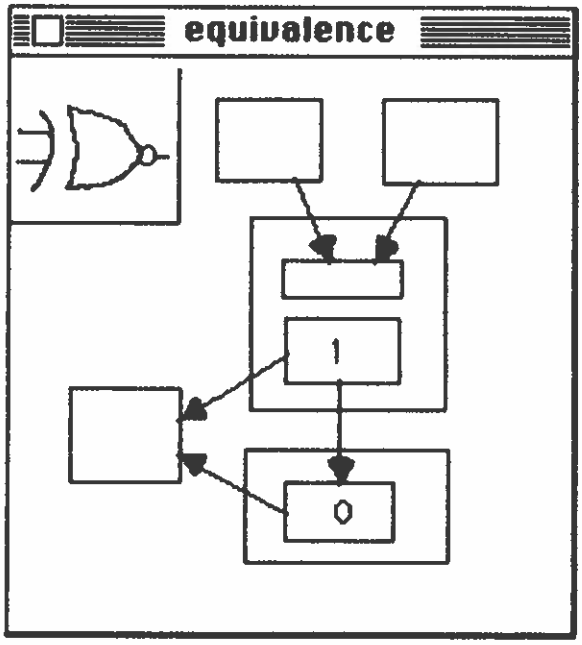
(e)



(f)

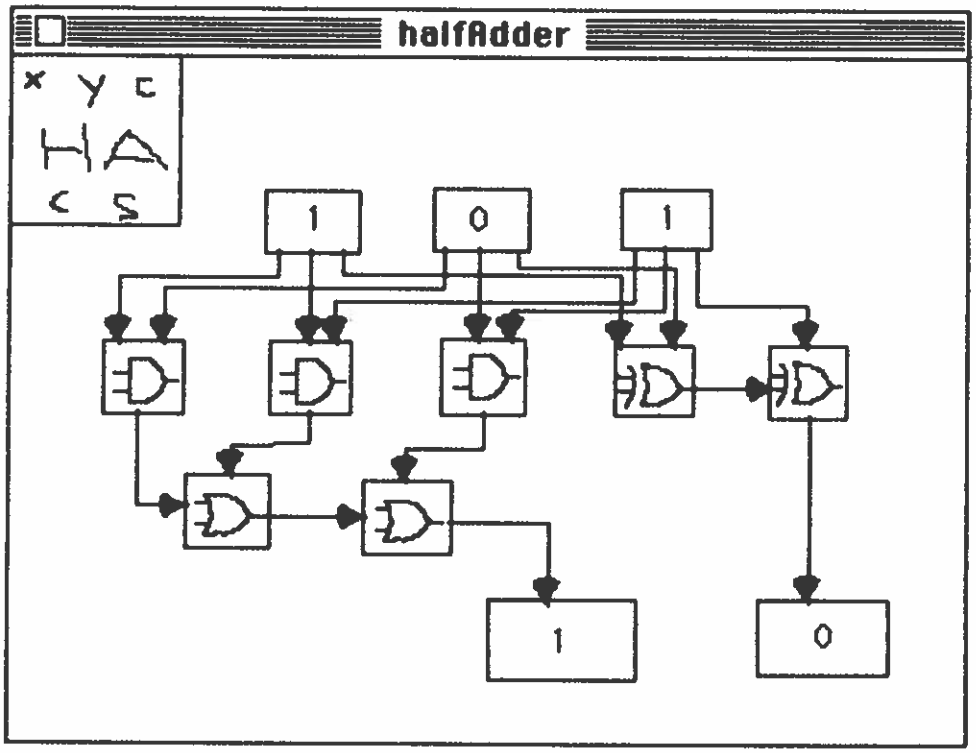


(g)

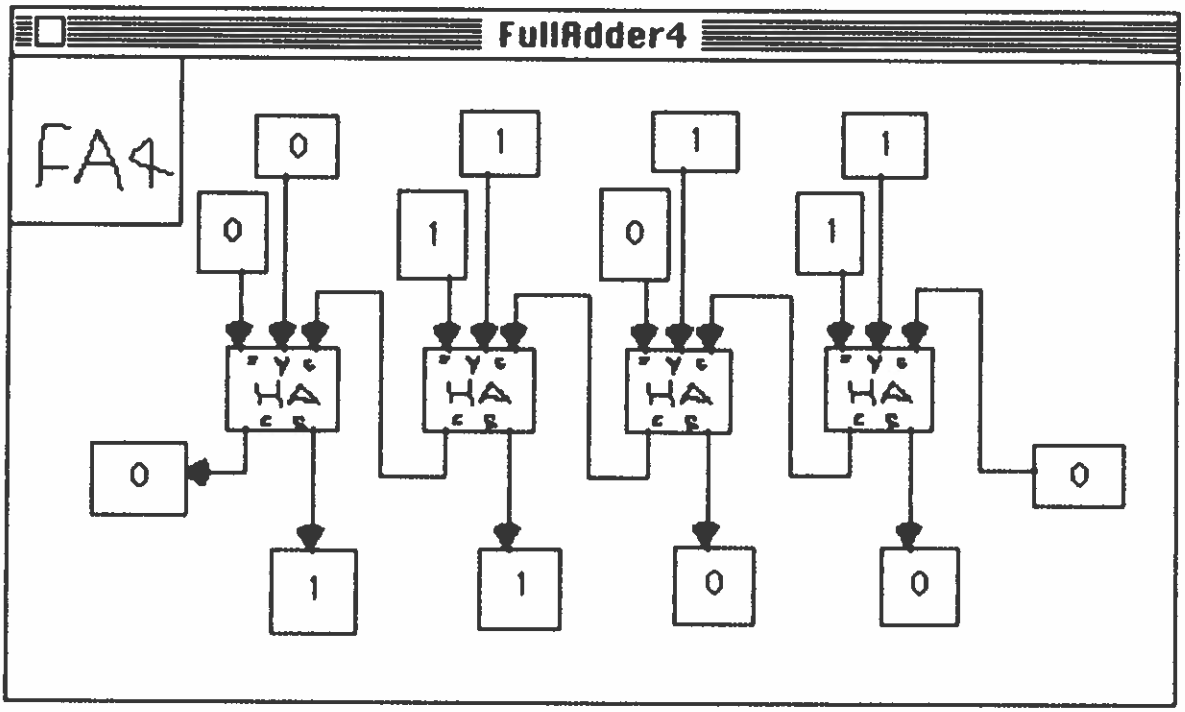


(h)

Figure 7.2: More Boolean Operations



(i)



(j)

Figure 7.3: Binary Adder

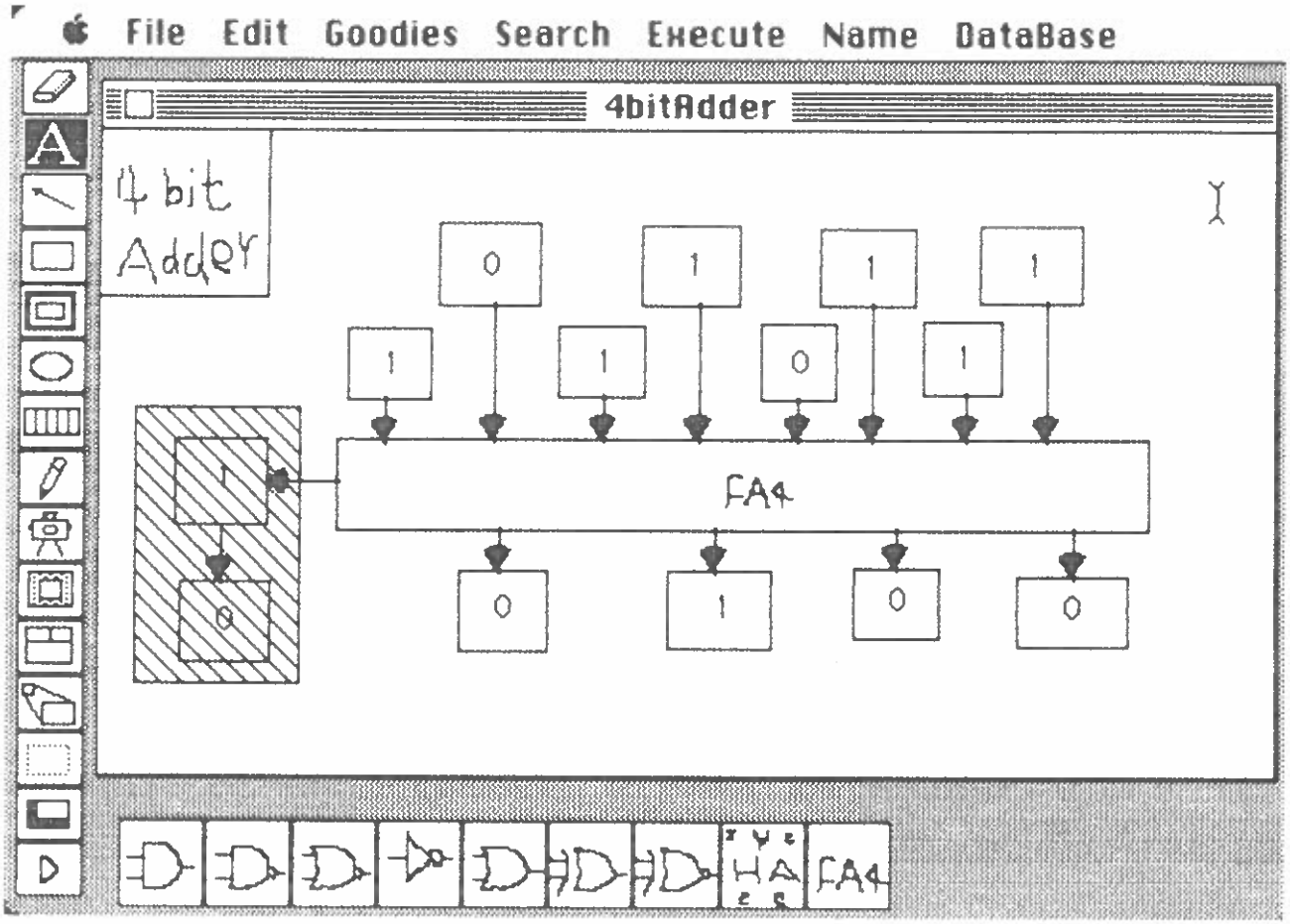


Figure 7.4: (k) 4 bits Adder in Mac STL

3. Definitions

In this section we will give a formal definition of HDM. The main purpose of formalization is to prove that HDM is determinate as a computation model, i.e., the outcome of a computation specified by a box-graph is unique, regardless how the computation is carried out. The determinancy will be shown in the next section.

The main idea in this formalization is the concept of elaboration which is an assignment of data values to the arrows in a box-graph. It is akin to the concept of model in the model theory of mathematical logic. Through this conceptual tool we can define HDM in a non-procedural way.

The following mathematical notations will be used.

$\subset$: is a subset of, (instead of 'is a proper subset of'),

$\equiv$: equal by definition,

$\langle = \rangle$: if and only if by definition,

$\#(S)$: the cardinarity of set S,

N: the set of natural numbers, {0, 1, 2, ... }.

A *hierarchical dataflow model,* HDM $\equiv$ (T, G), consists of a *data type* $T \equiv D \cup F \cup P$, and a set of *box-graphs* G, where

D is the set of *data values,*

$F \subset \bigcup_{m,n>0} (D^m \rightarrow D^n)$ is the set of *operations* on D,

$P \subset \bigcup_{m>0} (D^m \rightarrow 2)$ is the set of *predicates* on D.

A ***box-graph*** $G \equiv (B, B_0, A, s, d, \mu) \in \mathbf{G}$ is a 6-tuple such that

B is the totally ordered non-empty set of ***boxes***,

$B_0 \subset B$ is the set of ***open*** boxes,

($Bc \equiv B - Bo$ is the set of ***closed*** boxes,)

A is the set of ***arrows***,

$s : A \to B$ is the ***source*** of an arrow,

$d : A \to B$ is the ***destination*** of an arrow,

$\mu : B \to \mathbf{T} \cup \mathbf{G} \cup \{\phi\}$ is the ***content*** of a box,

satisfying the following conditions:

Note: A box $b \in B$ is ***empty*** if $\mu(b) = \phi$,

and b is ***complex*** if $\mu(b) \in \mathbf{G}$.

(1) (B, A, s, d) is a directed acyclic multigraph (damg), i.e.,

$(\forall \alpha \in A^*)((\forall i < |\alpha|)(s(a_{i+1}) = d(a_i)) \Rightarrow s(a_1) \neq d(a_{|\alpha|}))$,

where

α is a sequence of arrows,

$|\alpha|$ is the length of the sequence,

a_i is the i-th element of the sequence, $0 < i \leq |\alpha|$

(2) For each $b \in B$;

$in(b) \equiv d^{-1}(b) \subset A$

is a totally ordered set of ***incoming*** arrows of $b \in B$,

$out(b) \equiv s^{-1}(b) \subset A$

is a totally ordered set of ***outgoing*** arrows of $b \in B$.

Note that for different boxes orderings may be different. There may not be any total ordering on A. Also note that for any G, B is also a totally ordered set. We denotes these orderings by '<'. When the ordering is significant, the arrow sets can be considered as sequences of arrows;

$in(b) \equiv (a_1, a_2, \ldots, a_m)$,

$out(b) \equiv (\underline{a}_1, \underline{a}_2, \ldots, \underline{a}_n)$.

(3) The *arity* of non-empty box $b \in B$ is identical to the *arity* of its content $\mu(b) \in G$, i.e.,

$(\forall b \in B)(\mu(b) \neq \phi \Rightarrow arity(\mu(b)) = arity(b))$,

where $arity : B \cup T \cup G \rightarrow N \times N$ is defined by

$arity(b) \equiv (\#(in(b)), \#(out(b)))$ if $b \in B$,

$arity(x) \equiv (0, 0)$ if $x \in D$,

$arity(f) \equiv (m, n)$ if $f \in D^m \rightarrow D^n$,

$arity(p) \equiv (m, 0)$ if $p \in D^m \rightarrow 2$,

and

$arity(G) \equiv (\#(in(G)), \#(out(G)))$ if $G \in G$,

where

$in(G) \equiv \{ b \in B \mid \mu(b) = \phi \wedge in(b) = \phi \}$ are the *in-boxes* of G, and

$out(G) \equiv \{ b \in B \mid \mu(b) = \phi \wedge out(b) = \phi \}$ are the *out-boxes* of G.

(4) There exists a *binding rule* consisting of two bijections, **f** and **g**, for each complex box $b \in B$, where $\mu(b) \in G$, such that

$$\mathbf{f}: in(b) \leftrightarrow in(\mu(b))$$

$$\mathbf{g}: out(b) \leftrightarrow out(\mu(b))$$

and for any $b, b' \in B$,

$$b < b' \text{ if and only if } \mathbf{f}(b) < \mathbf{f}(b') \wedge \mathbf{g}(b) < \mathbf{g}(b').$$

In order to define the notion of consistency in HDM, we introduce the semantic concept of elaboration, which is an assignment of data values to the set of arrows. Given an elaboration of a box-graph, the data values carried by the incoming and outgoing arrows on an individual box should satisfy the local consistency conditions imposed by the content of the box. A box-graph $G \in \mathbf{G}$ is defined as consistent if there exists an elaboration for G, otherwise inconsistent.

A mapping $\theta : A \rightarrow D \cup \{\phi\}$ is an *elaboration* of $G \equiv (B, B_o, A, s, d, \mu) \in \mathbf{G}$, if for any $b \in B$ the following conditions hold:

(1) When $\mu(b) = \phi$;

If there exists one arrow, a, of b whose value is not null ($\neq\phi$) under θ, then at least one incoming arrow must heve non-null value, and all outgoing arrows and all non-null incoming arrows must have the same value $\theta(a)$, i.e.,

$$(\forall a \in in(b) \cup out(b))(\theta(a) = \phi) \vee$$

$$(\exists a \in in(b))(\theta(a) \neq \phi \wedge$$

$$(\forall \underline{a} \in in(b))(\theta(\underline{a}) \neq \phi \Rightarrow \theta(\underline{a}) = \theta(a)) \wedge$$

$$(\forall \underline{a} \in out(b))(\theta(\underline{a}) = \theta(a))).$$

(2) When $\mu(b) \in D$;

All outgoing arrows and all non-null incoming arrows must have the same value $\mu(b)$, i.e.,

$$(\forall a \in in(b))(\theta(a) \neq \phi \Rightarrow \theta(a) = \mu(b)) \wedge (\forall a \in out(b))(\theta(a) = \mu(b)).$$

(3) When $\mu(b) \in F$, and $\mu(b) : D^m \rightarrow D^n$ for some $m,n > 0$;

The values on the outgoing arrows are the values of the function $\mu(b)$ evaluated with the values on the incoming arrows as its arguments, i.e.,

$\theta(out(b)) = \mu(b)\,(\theta(in(b)))$, which means by convention

$(\theta(\underline{a}_1), \theta(\underline{a}_2), \ldots, \theta(\underline{a}_n)) = \mu(b)\,(\theta(a_1), \theta(a_2), \ldots, \theta(a_m))$,

where

$out(b) \equiv (\underline{a}_1, \underline{a}_2, \ldots, \underline{a}_n)$, $in(b) \equiv (a_1, a_2, \ldots, a_m)$.

Note that the arity of b is same as the arity of $\mu(b)$ by the definition of the box-graph, and that out(b) and in(b) are totally ordered sets. This condition implies that all incoming arrows have non-null values.

(4) When $\mu(b) \in P$, and $\mu(b) : D^m \rightarrow 2$ for some $m > 0$;

The predicate $\mu(b)$ is true for the arguments carried by the incoming arrows, i.e.,

$\mu(b)\,(\theta(in(b)))$, which means

$\mu(b)\,(\theta(a_1), \theta(a_2), \ldots, \theta(a_m))$.

Note that there are no outgoing arrows for b, and that all incoming arrows have non-null values.

(5) When $\mu(b) \in \mathbf{G}$, and $\mu(b) = \mathbf{G} = (\mathbf{B}, \mathbf{B_c}, \mathbf{A}, \mathbf{s}, \mathbf{d}, \mu) \in \mathbf{G}$;

(5.1) When $b \in B_c$;

Let $\mathbf{G'}$ be the result of binding the incoming values of b with the in-boxes of $\mathbf{G}$ and the outgoing values of b with the out-boxes of $\mathbf{G}$, i.e., the in-boxes of $\mathbf{G}$ are filled with the values of the incoming arrows of b under θ and the out-boxes of $\mathbf{G}$ are filled with the values of the outgoing arrows of b, according to the binding rule. Then the condition is that if there exists no elaboration of $\mathbf{G'}$, then all outgoing arrows of b must have the null value.

Formally,

$$(\exists a \in \mathrm{out}(b))(\theta(a) \neq \phi) \Rightarrow (\exists \underline{\theta})(\ \underline{\theta} \text{ is an elaboration of } \mathbf{G'}\ )$$

where

$$\mathbf{G'} = (\mathbf{B}, \mathbf{B_c}, \mathbf{A}, \mathbf{s}, \mathbf{d}, \mu') \in \mathbf{G}, \quad \text{and}$$

$$\mu'(\mathbf{b}) \equiv \theta(\mathbf{f}^{-1}(\mathbf{b})) \qquad \text{if } \mathbf{b} \in \mathrm{in}(\mathbf{G}),$$

$$\equiv \theta(\mathbf{g}^{-1}(\mathbf{b})) \qquad \text{if } \mathbf{b} \in \mathrm{out}(\mathbf{G}),$$

$$\equiv \mu(\mathbf{b}) \qquad \text{otherwise.}$$

Note that $\mathbf{f}$ and $\mathbf{g}$ are the binding rule justified by the definition of box-graph condition (4).

(5.2) When $b \in B_o$;

There exists an elaboration of $\mathbf{G'}$, i.e.,

$$(\exists \underline{\theta})(\ \underline{\theta} \text{ is an elaboration of } \mathbf{G'}\ ).$$

A box-graph $G \in \mathbf{G}$ is *consistent* if there exists an elaboration of G, otherwise *inconsistent*. G is *solvable* if G is consistent and has no in-boxes. G is *solved* if G is consistent and has no empty boxes.

Using the following notation for the consistency,

$$Cons(G) \equiv G \text{ is consistent,} \qquad \text{where } G \in \mathbf{G},$$

the above definitions can be expressed by:

$$G \text{ is } \textit{solvable} \iff Cons(G) \wedge in(G) = \phi,$$

$$G \text{ is } \textit{solved} \iff Cons(G) \wedge \mu^{-1}(\phi) = \phi.$$

The conditions for (5.1) and (5.2) also can be rewritten as:

(5.1) $(\exists a \in out(b))(\theta(a) \neq \phi) \Rightarrow Cons(\mathbf{G}')$,

(5.2) $Cons(\mathbf{G}')$.

Finally, we will define the notion of <u>solution</u> for a solvable box-graph. In the next section we will show that the solution is unique by showing that if a box-graph is solvable, there is a unique elaboration for the graph.

The set of empty boxes of G is called the ***base*** of G, denoted by:

$$base(G) \equiv \{ b \in B \mid \mu(b) = \phi \}.$$

When G is solvable, an assignment of data values to the base of G is called a solution for G if the completed box-graph is consistent. Formally,

$$\sigma : base(G) \to D \text{ is a } \textit{solution} \text{ for } G \iff Cons(G_\sigma),$$

where

$$G_\sigma \equiv (B, B_o, A, s, d, \mu_\sigma) \in \mathbf{G},$$

and

$$\begin{cases} \mu_\sigma(b) \equiv \sigma(b) & \text{if } b \in base(G), \\ \equiv \mu(b) & \text{otherwise.} \end{cases}$$

4. Determinancy

In the previous sections we defined the concepts of consistency, solvability, and solution in HDM without involving any procedural concepts. There remain two issues we have to address before we construct a programming language based on HDM: one is to show that HDM is determinate, i.e., if a box-graph G is solvable, then there exists only one solution, and the other is to construct an algorithm by which the unique solution can be derived. In this section we will prove the following theorem that an elaboration of a box-graph is unique.

Theorem: Let $G \equiv (B, B_o, A, s, d, \mu) \in \mathbf{G}$ be a box-graph.

If $G \in \mathbf{G}$ is solvable, then the elaboration is unique, i.e.,

if $\theta_1, \theta_2 : A \to D \cup \{ \phi \}$ are elaborations of G, then

$\theta_1(a) = \theta_2(a)$ for any $a \in A$.

The main tool for the proof will be the principle of *structural induction* for well-founded sets[6]. A partially ordered set is *well-founded* if there is no infinite sequence of decreasing elements. Every well-ordered set is well-founded. Every finite poset is well-founded. Let $P(x)$ be an arbitrary predicate on a well-founded poset $(X, \leqq)$. The principle of structural induction states that: To show that $(\forall x \in X)\, P(x)$, it is sufficient to show that $(\forall x \in X)(((\forall y \leqq x)\, P(y)) \Rightarrow P(x))$. Note that the condition implies $P(x)$ for each minimal element x of X.

Before we prove the theorem, we need notational conventions and one lemma stating that if a box-graph is consistent, then filling each empty box with the non-null value of the elaboration carried by one incoming arrow will not change the consistency property.

Notations: (1) Let θ be an elaboration of G. We extend θ to the sequences of arrows as follows:

$$\theta(a_1, a_2, \ldots, a_m) \equiv (\theta(a_1), \theta(a_2), \ldots, \theta(a_m)), \quad a_i \in A.$$

(2) $G/\mu = (B, B_0, A, s, d, \mu)$

where μ is an arbitrary content function on B.

Lemma:

(I) If θ is an elaboration of G, then it is also an elaboration of G/μ, where

$\mu(b) \equiv \theta(a)$ if $b \in \text{base}(G)$ and $\theta(a) \neq \phi$ for some $a \in \text{in}(b)$,

$\equiv \mu(b)$ otherwise.

(II) If θ is an elaboration of G, then it is also an elaboration of G/μ', where

$\mu'(b) \equiv \phi$ if $\mu(b) \in D$ and $\theta(a) \neq \phi$ for some $a \in \text{in}(b)$,

$\equiv \mu(b)$ otherwise.

Proof: (I) We want to show that for each box of G the conditions given in the definition of elaboration are satisfied by θ after the content of an empty box is changed. By the definition of G/μ, we must consider only those empty boxes which has at least one incoming arrow with non-null value under θ. Let $b \in B$ be one of such boxes, and let $a \in \text{in}(b)$ be such that $\theta(a) \equiv d \neq \phi$. By the condition (1), all non-null incoming arrows and all outgoing arrows of b has the value d under θ. By the definition of $\mu(b)$, $\mu(b) = d$. Therefore, the condition (2) is also satisfied by θ for G/μ.

(II) Similarly, the condition (2) implies the condition (1).

Q.E.D.

Note that the above lemma indicates how the solution for a solvable box-graph can be obtained in simple steps, once the elaboration of the graph is found.

Proof of the Theorem: We must use two induction principles in this proof; the mathematical induction on the levels of nesting box-graphs, and the structural induction on the partial ordered set of boxes in a box-graph. Since our application of the mathematical induction is straightfoward, we omit the base step of the mathematical induction. The inductive hypothesis is that the theorem holds for any box-graph with less number of nesting.

Since (B, A, s, d) is a directed acyclic multigraph, (B,<) is a partially ordered set with the following ordering:

$b_1 < b_2 \Leftrightarrow (\exists \alpha \in A^*)(s(a_1) = b_1 \wedge d(a_{|\alpha|}) = b_2 \wedge (\forall i < |\alpha|)(s(a_{i+1}) = d(a_i)))$.

Furthermore, B is a well-founded set because B is finite.

Assume that G is solvable and θ_1 and θ_2 be elaborations of G. We will show by structural induction that:

$$(\forall b \in B)\ P(b)$$

where $P(b) \equiv (\theta_1(in(b)) = \theta_2(in(b)) \Rightarrow \theta_1(out(b)) = \theta_2(out(b)))$.

Let $b \in B$ be an arbitrary box in G, and assume that P(b') for any $b' < b$. We want to show that P(b).

Base Step: b is a minimal element of B, i.e., $in(b) = \phi$.

Since G is solvable, there are three cases for the content of b:

(1) $\mu(b) \in D$.

By definition of *elaboration*, $\theta_1(out(b)) = \theta_2(out(b)) = \mu(b)$. $\therefore P(b)$.

(2) $\mu(b) \equiv \mathbf{G} \in \mathbf{G}$ and $b \in B_c$.

We want to show that $\theta_1(a) = \theta_2(a)$ for any $a \in$ out(b).

Since in(**G**) = ϕ, **G** is solvable if and only if **G** is consistent.

(2.1) when **G** is consistent with an elaboration θ.

Let $\mathbf{G'}_1$ and $\mathbf{G'}_2$ be the result of binding the outgoing values of b, based on θ_1 and θ_2 respectively, with the out-boxes of **G**. Then $\mathbf{G'}_1$ and $\mathbf{G'}_2$ must be consistent with some elaborations θ_1 and θ_2, because if $\mathbf{G'}_1$ is not consistent then $\theta_1(a) = \phi$ for all $a \in$ out(b) by the definition condition (5), and $\mathbf{G} = \mathbf{G'}_1$, contradicting with the assumption that **G** is consistent.

Then, for some $\mathbf{a} \in$ in(g(a)),

$\theta_1(\mathbf{a}) = \theta_1(a)$

$\theta_2(\mathbf{a}) = \theta_2(a)$ by the definition condition (2), and

θ_1 and θ_2 must be an elaboration of **G** by Lemma(II).

However, by the inductive hypothesis **G**'s elaboration must be unique, i.e., $\theta(\mathbf{a}) = \theta_1(\mathbf{a}) = \theta_2(\mathbf{a})$, therefore $\theta_1(a) = \theta_2(a)$. $\therefore$ P(b).

(2.2) when **G** is not consistent.

By the definition condition (5), $\theta_1(a) = \theta_2(a) = \phi$.

(3) $\mu(b) \equiv \mathbf{G} \in \mathbf{G}$ and $b \in B_o$.

If **G** is inconsistent, by definition of open box, G must be inconsistent.

Since G is consistent by the assumption, **G** must be consistent.

The same argument as (2.1) above holds here.

End of Base Step.

Induction Step: Let $b \in B$ where $in(b) \neq \phi$, and assume that $\theta_1(in(b)) = \theta_2(in(b))$. We want to show that $\theta_1(out(b)) = \theta_2(out(b))$, i.e., for any $a \in out(b)$, $\theta_1(a) = \theta_2(a)$.

There are five cases to consider for the content of b:

(1) $\mu(b) = \phi$.

If $\theta_1(in(b)) = \theta_2(in(b)) = \phi$, then $\theta_1(out(b)) = \theta_2(out(b)) = \phi$.

If for some $\underline{a} \in in(b)$, $\theta_1(\underline{a}) \neq \phi$, then $\theta_1(\underline{a}) = \theta_1(a)$ for any $a \in out(b)$.

Similarly, $\theta_2(\underline{a}) = \theta_2(a)$ for any $a \in out(b)$.

By the assumption, $\theta_1(\underline{a}) = \theta_2(\underline{a})$, therefore $\theta_1(a) = \theta_2(a)$.

(2) $\mu(b) \in D$.

By the definition condition (2), $\theta_1(out(b)) = \theta_2(out(b)) = \mu(b)$.

(3) $\mu(b) \in F$.

Since $\theta_1(in(b)) = \theta_2(in(b))$,

$\theta_1(out(b) = \mu(b)(\theta_1(in(b))) = \mu(b)(\theta_2(in(b))) = \theta_2(out(b))$.

(4) $\mu(b) \in P$.

Vacuously true.

(5) $\mu(b) = \mathbf{G} \in \mathbf{G}$ and $b \in B_c$.

Let $\mathbf{G}'$ be the result of binding the incoming values of b, $\theta_1(in(b)) = \theta_2(in(b))$ with the in-boxes of $\mathbf{G}$. Then, by the same argument as in (2.1) of the base step, with $\mathbf{G}'$ in place of $\mathbf{G}$ in (2.1), we can conclude $\theta_1(out(b)) = \theta_2(out(b))$.

(6) $\mu(b) = \mathbf{G} \in \mathbf{G}$ and $b \in B_o$.

Similarly to (5).

Therefore, by the principle of structural induction,

$(\forall b \in B)\ (\ \theta_1(in(b)) = \theta_2\ (in(b)) \Rightarrow \theta_1(out(b)) = \theta_2\ (out(b))\ )$.

Since for any $a \in A$, $a \in out(b)$ for some $b \in B$; $\theta_1(a) = \theta_2(a)$ for any $a \in A$.

End of Proof for Theorem.

5. Conclusions

The hierarchical dataflow model is only a part of the Show & Tell Language (STL) semantics. Data query operations, recursion, iterations, parameterizations are available in STL, but they are not included in HDM to keep the model simple enough so that it can be compared with the traditional dataflow model. However, HDM is an important part of our attempts to formalize an icon driven programming language such as STL. A formal syntax for STL has not been developed yet.

The working hypothesis for the Show & Tell Project is that for school children dataflow concept is intuitively easy to understand. Recently, STL on Macintosh was used to test the hypothesis in the local school environment, with 4-th and 7-th grade students. Preliminary indication from the test data is that the hypothesis is a valid one. A detailed analysis of the test will be described in a separate report.

6. References:

1. Glinert, E. P. PICT: Experiments in the Design of Interactive, Graphical Programming Environments. Technical Report 85-01-01, Department of Computer Science, University of Washington, Seatle, WA., 1985.
2. Kimura, T. D. Show and Tell: A Programming Language for School Children. Technical Report WUCS-85-6, Department of Computer Science, Washington University, St. Louis, MO., May1985.
3. Davis, A. L., and Keller, R. M. Data Flow Program Graphs. IEEE *Computer*, vol. 15, No. 2, February 1982, pp. 26-41.
4. Shneiderman, B. Direct Manipulation: A Step Beyond Programming Languages. IEEE *Computer*, vol 16, No. 8, August 1983, pp. 57-69.
5. Kimura, T. D. Completion Problem and Its Solution for Context-Free Languages (Algebraic Approach). Moore School Report 72-09, University of Pennsylvania, Philadelphia, PA., May 1971.
6. Levy, L. L. *Discrete Structures of Computer Science.* John Wiley & Sons, 1980.